Advantage™ CA-Easytrieve® Plus Report Generator

SQL Interface Option Guide

6.4

Computer Associates®

Contents

Chapter 3: Library Section Definition

Chapter 4: Automatic Processing

Chapter 5: Controlled Processing

Chapter 1

Overview

This guide explains the optional product facilities that interface with the following SQL databases:

- IBM DB2 for OS/390 and z/OS
- IBM DB2 for VSE (SQL/DS)
- Jasmine CA-Datacom/DB for SQL
- Jasmine CA-IDMS SQL
- ORACLE

Before CA-Easytrieve Plus can process these databases, the Jasmine CA-Pan/SQL Interface product, version 2.4, must be correctly installed. See the *CA-Pan/SQL SQL Interface Getting Started* for complete information.

To use this facility effectively, you should have a basic knowledge of SQL and the given database management system to be processed.

Topics

This guide discusses the following topics:

- Working environments for SQL
- Host variable definitions
- Automatic table processing
- Native SQL commands and the GET command as used in controlled processing
- JCL

Related Publications

The following publication, produced by Computer Associates, is either referenced in this documentation or is recommended reading:

- *CA-Pan/SQL SQL Interface Getting Started*

The following publications, not produced by Computer Associates, are either referenced in this documentation or are recommended reading:

- *ORACLE Installation and System Administration Guide*
- *IBM DB2 Reference Manual*

Revision Summary for DB2 for OS/390 and z/OS Elements

The SQL interface for has the following enhancements:

1. Support of DSNALI Above the Line

   Release 4.1 of DB2 for OS/390 and z/OS linked DSNALI as AMODE 31. This caused DSNALI to be loaded Above The Line.

2. Validation of Host Variable Datatypes at Compile Time
3. Support of CALL, ALLOCATE, and ASSOCIATE statements for STATIC-ONLY processing..

   The CALL procedure, ALLOCATE cursor, and ASSOCIATE LOCATORS statements are now supported by PanSQL for a BIND option of STATIC-ONLY.

4. Support specification of a PLAN and SSID at execution time

   You can specify an execution time PLAN and SSID with a PAN$SQL DD file. This enables you to run with a different planname on different subsystems.

5. Support for Execution of Statically Bound Applications under the TSO Terminal Monitor Program

   This enables your called subroutines to be linked with DSNELI for both Jasmine CA-Easytrieve applications and other language applications.

Programming Methods

There are two programming methods supported for processing SQL databases:

- Using native SQL statements to manually manage the SQL cursor.
- Permitting CA-Easytrieve Plus to automatically manage the SQL cursor.

Native SQL Statements

CA-Easytrieve Plus supports most of the SQL statements available for a given DBMS. The exceptions are those statements that are compiler directives and statements that cannot be dynamically *prepared.* Using these native SQL statements, you can code fully SQL-compliant programs in which you control the SQL cursor operation. All native SQL statements are prefixed with the *SQL* keyword. A list of all supported and unsupported SQL commands is provided in the section called Usage Notes.

Automatic Cursor Management

There are two ways that CA-Easytrieve Plus can manage the SQL cursor for you:

- CA-Easytrieve Plus files
- Automatic retrieval without a file

SQL Statement Rules

There are several differences in the rules when coding SQL control statements in CA-Easytrieve Plus. The following syntax rules apply:

- Operators must be separated by blanks.
- Standard CA-Easytrieve Plus continuation conventions are followed.
- Commas are **not** ignored.
- The period is used as a qualification separator, not to signify end-of-statement.
- The colon is used to identify host/indicator variables, not as a qualification separator.
- An SQL statement cannot be followed by another statement on the same source record.

Chapter

2

Program Environment

CA-Easytrieve Plus processes SQL statements using the CA-Pan/SQL Interface product. A specific implementation exists for each supported database:

- For DB2 for OS/390 and z/OS, a dynamic and static interface are supported.
- For DB2 for VSE, *extended dynamic* SQL is supported. SQL statements are dynamically prepared during the compilation of your CA-Easytrieve Plus program and an access module or package is created. At runtime, SQL statements are executed from the access module or package.
- The CA-Datacom/DB SQL interface is very similar to the DB2 for VSE interface. An access plan is created at compile time from which SQL statements are executed.
- The CA-IDMS SQL interface is strictly a dynamic interface for both compilation and execution.
- The ORACLE interface is strictly a dynamic interface for both compilation and execution.

Units of Work

Each CA-Easytrieve Plus activity is considered a separate SQL unit of work.

A COMMIT is executed following each JOB activity that contains SQL statements. A ROLLBACK is automatically executed if an error condition is detected or if you code a STOP EXECUTE statement in your program.

PARM Statement Parameters

The following PARM statement parameters set the SQL environment for the program:

For DB2 for OS/390 and z/OS:

- SQLID
- SSID
- PLAN
- BIND
- SQLSYNTAX.

For DB2 for VSE:

- USERID
- PREPNAME
- SQLSYNTAX.

For CA-Datacom/DB:

- PLANOPTS
- PREPNAME
- SQLSYNTAX.

For ORACLE:

- USERID
- SSID
- SQLSYNTAX.

For CA-IDMS:

- USERID
- SQLSYNTAX.

All SQL Environments

In all environments, use the SQLSYNTAX parameter to specify the level of SQL syntax checking to perform on the SQL statements in your program.

SQLSYNTAX FULL

SQLSYNTAX FULL indicates that SQL statements undergo detail-level syntax checking. An SQL PREPARE statement executes for those SQL statements that can be dynamically prepared. Your DBMS must be available to CA-Easytrieve Plus.

SQLSYNTAX PARTIAL

SQLSYNTAX PARTIAL checks your SQL statements for valid commands and secondary keywords. No connection is made to your DBMS unless you coded an SQL INCLUDE statement in your program. Your program cannot execute until it undergoes FULL syntax checking.

SQLSYNTAX NONE

You can use the SQLSYNTAX NONE parameter on the PARM statement with a static bind if you want the DB2 for OS/390 and z/OS preprocessor to perform syntax checking in a batch environment. An option of NONE performs partial syntax checking on your program. If no partial level compile errors are found and a BIND option of STATIC-ONLY is specified and no other non-SQL syntax errors are found, your program continues to execution.

You can use SQLSYNTAX NONE with BIND STATIC-ONLY to process unqualified tables. With SQLSYNTAX NONE, you eliminate the dynamic bind which qualifies all unqualified tables. You can then set a qualifier when you BIND the DBRM of the static-command-program.

DB2 for OS/390 and z/OS

SQLID

The SQLID parameter of the PARM statement results in the SQL interface executing the SQL SET CURRENT SQLID command at compile time. The SQL SET CURRENT SQLID command is executed at runtime for automatic processing. Execution of the SQL SET CURRENT SQLID command is valid for sites that have an external security package that supports group IDS. It sets the qualification of unqualified tables for dynamic SQL processing. See Qualifying DB2 for OS/390 and z/OS Tables later in this chapter for more information.

It is recommended to use SQLSYNTAX NONE with BIND STATIC-ONLY rather than SQLID to set the qualification for tables.

SSID Parameter

The SSID parameter of the PARM statement can explicitly specify the DB2 for OS/390 and z/OS subsystem. If the SSID parameter is not coded, the SQL interface gets the DB2 for OS/390 and z/OS subsystem ID from the DB2 for OS/390 and z/OS system default module DSNHDECP. DSNHDECP is made available through the JOBLIB or STEPLIB libraries.

The valid format is *xxxx/yyyyyyyy*, where *xxxx* is a valid DB2 for OS/390 and z/OS subsystem ID and *yyyyyyyy* is a valid DB2 for OS/390 and z/OS location ID. Neither value is required. You can specify either one. If you specify a location ID, you must precede it with a slash, such as SSID=/yyyyyyyy.

Static SQL

Static SQL improves the performance of an SQL program. In a static SQL program, all SQL statements are known ahead of time and an optimized plan is created before execution time.

Static SQL is specified by two parameters on the PARM statement, PLAN and BIND.

- A PLAN parameter specifies the name of the DB2 for OS/390 and z/OS static-command-program and its planname. The command program is linked as a separate load module.
- A BIND parameter of STATIC-ONLY or ANY generates the static-command-program. Execution can be static or dynamic.

To execute your program statically, you must run special steps to create and link the static command program and plan. See the "Executing Your SQL Program" chapter for more information.

DB2 for VSE

USERID

Specify the DB2 for VSE user ID and password to use for compiling the program on the USERID parameter of the PARM statement.

- For automatic processing, CA-Easytrieve Plus executes a CONNECT using the same parameters.
- For controlled processing, you must code a CONNECT statement to be the first SQL statement the program executes.

PREPNAME

Specify the name of the DB2 for VSE access module for this program on the PREPNAME parameter of the PARM statement. When an SQL program is compiled, an access module is created or replaced. You should use unique access module names for each application program to avoid using the default access module.

In a high volume system, using the default PREPNAME can result in catalog contention and a -911 SQLCODE resulting from a rollback.

SSID

Provide a database ID on the SSID parameter to connect to a non-default database.

The valid format is *yyyyyyyy*, where *yyyyyyyy* is a valid database ID.

CA-Datacom/DB

PLANOPTS

Use the PLANOPTS parameter on the PARM statement to specify the name of a CA-Pan/SQL plan options module to override the default plan module DQSMPLN@. See the *CA-Pan/SQL SQL Interface Getting Started* for more information about defining your own options module.

PREPNAME

Specify the name of the plan and authorization ID to create the SQL plan for CA-Datacom/DB. The plan is created using the options defined by the CA-Pan/SQL plan options module.

ORACLE

USERID

Specify the user ID to use for compiling the ORACLE program on the USERID parameter of the PARM statement. For more details on the USERID parameter, refer to the “System Overview” chapter of the *Reference Guide.*

SSID

- The SSID parameter of the PARM statement can specify the appropriate subsystem. Refer to the “System Overview” chapter and the “Options Table” appendix in the *Reference Guide* for more details on the SSID parameter.
- You can also specify the subsystem through an ORA@ssid DD statement. Refer to your *ORACLE Installation and System Administration Guide* for more details on using the ORA@ssid DD statement.

CA-IDMS

USERID

Specify the user ID to use for compiling the CA-IDMS program on the USERID parameter of the PARM statement. For more details on the USERID parameter, refer to the “System Overview” chapter of the *Reference Guide.*

Qualifying DB2 for OS/390 and z/OS Tables

Unqualified tables are implicitly qualified by the primary authorization ID of the program. This ID is usually established by the USER parameter of your JCL JOB card. You can modify qualification in one of three ways:

- SQLID keyword on the CA-Easytrieve PARM statement
- SQL SET CURRENT SQLID command
- OWNER or QUALIFIER parameter on the DB2 for OS/390 and z/OS BIND statement.

Use of the SQLID and its affect on table qualification depends on whether automatic or controlled processing is performed and if STATIC or DYNAMIC SQL is used.

To eliminate the authorization problems encountered with the use of SQLID, use SQLSYNTAX NONE with BIND STATIC-ONLY. This enables the use of unqualified table names and bypasses the dynamic prepare of SQL statements. Unqualified table names can then be qualified by the BIND process.

Qualification with Automatic Processing

DYNAMIC SQL

Use the SQLID on the PARM statement to set the table qualification. SQLID sets the qualification at compile time when table validation is performed. Also, it executes a SET CURRENT SQLID at runtime.

The primary authorization ID must have the proper DB2 for OS/390 and z/OS authority to issue the SET CURRENT SQLID command. This authority is established through the implementation of an external security system.

In the following example, the use of SQLID sets the current SQLID to ‘SYSIBM' during compilation, thus DB2 for OS/390 and z/OS verifies that the table called ‘SYSIBM.SYSPLAN' exists.

A SET CURRENT SQLID command automatically executes at runtime to establish the qualification of SYSIBM for table SYSPLAN.

```
PARM SQLID('SYSIBM')
DEFINE WKNAME       W   18  A
DEFINE WKCREATOR    W   8   A
JOB INPUT SQL
  SELECT NAME, CREATOR FROM  SYSPLAN   ORDER  BY NAME            +
                                       INTO  :WKNAME,  :WKCREATOR
```

STATIC SQL

To modify for static execution, you change the PARM statement as follows:

```
PARM PLAN  (TESTPLAN TESTPGM)  LINK  (EZTPGM R)     +
     BIND STATIC-ONLY SQLID('SYSIBM')
```

The SQLID acts as it does for dynamic processing during the compilation of the program. It executes a SET CURRENT SQLID = 'SYSIBM'.

Since we are using automatic processing, the SET CURRENT SQLID command is generated for runtime execution. However, since we are running static SQL, the SET CURRENT SQLID has no affect on the qualification of tables. The OWNER and QUALIFIER parameters of the BIND process establish the qualification of unqualified tables. Refer to your *IBM DB2 Reference Manual* for more information on the SET and BIND commands.

For both STATIC and DYNAMIC processing, the primary authorization ID of the compile or runtime process must have the authority to execute the SET CURRENT SQLID command. Invalid authorization results in an SQLCODE of -553.

Qualification with Controlled Processing

DYNAMIC SQL

Consider the partial program in the following example. The SQLID on the PARM statement generates a SET CURRENT SQLID = 'SYSIBM' during the compilation process. Thus, the table 'SYSIBM.SYSPLAN' is validated.

During the execution phase, the SET CURRENT SQLID = 'SYSIBM' is the first SQL statement executed, so the DECLARE CURSOR and the FETCH is for the table 'SYSIBM.SYSPLAN'.

If you did not code the SET CURRENT SQLID command in your program, the DECLARE CURSOR is executed for the table *userid*.SYSPLAN and a -514 SQLCODE occurs, since this table does not exist.

```
PARM SQLID('SYSIBM')
DEFINE WKNAME        W    18 A
DEFINE WKCREATOR     W    8  A
SQL   DECLARE C1 CURSOR  FOR             +
      SELECT  NAME, CREATOR              +
      FROM    SYSPLAN                    +
      ORDER BY NAME
JOB INPUT NULL
SQL SET CURRENT SQLID='SYSIBM'
  SQL OPEN C1
  PERFORM CHECK-SQL-CODE
  DO WHILE SQLCODE EQ 0
    SQL  FETCH C1 INTO :WKNAME,:WKCREATOR
    PERFORM CHECK-SQL-CODE
    IF SQLCODE = 0
       PRINT REPORT1
    END-IF
  END-DO
  SQL CLOSE C1
  PERFORM CHECK-SQL-CODE
STOP
```

STATIC SQL

To modify for Static SQL processing, you change the PARM statement to look like:

```
PARM PLAN (TESTPLAN TESTPGM) LINK (EZTPGM R)     +
     BIND STATIC-ONLY SQLID('SYSIBM')
```

You also remove the SET CURRENT SQLID = ‘SYSIBM' statement. The SET CURRENT command is a dynamic only command and thus has no affect on a STATIC program.

With these modifications, the SQLID on the PARM statement still generates the SET CURRENT SQLID command during compilation and then validates the table SYSIBM.SYSPLAN.

To qualify the tables at execution, however, you must use the appropriate parameters, OWNER and QUALIFIER, on the BIND statement. A sample BIND statement is shown below. Refer to your *IBM DB2 Reference Manual* for information on the DB2 for OS/390 and z/OS Bind statement.

```
BIND PLAN (TESTPLAN) MEMBER (TESTDBRM) VALIDATE (RUN) -
     ACT(REPLACE) ISOLATION (CS) OWNER(TESTUSER) QUALIFIER(SYSIBM)
```

SQL Error Handling

At CA-Easytrieve Plus compilation time, the SQL statements in your program undergo detail syntax checking by the underlying database management system unless SQLSYNTAX=NONE is coded. Detail syntax checking means that the SQL statement is processed by an SQL PREPARE statement that the SQL interface manages. If the database detects an error, the SQL interface returns error information with an appropriate error message.

At runtime, CA-Easytrieve Plus checks the SQLCODE in the SQLCA for you and issues an appropriate error message if an error was detected during automatic processing. It is the user's responsibility to check the SQLCODE after each SQL statement to determine its success or failure when using Native SQL. The user can display the various SQLCA fields and obtain the information to determine the cause of any SQL error.

SQL Error Message Format

The format of the message containing the SQLCA error information returned to the program, varies with the underlying database management system. The format for each SQL database is shown below.

DB2 for OS/390 and z/OS, DB2 for VSE, and ORACLE

```
SQLCODE FROM SQLCA IS sqlcode
```

CA-Datacom

```
SQL ERROR, SQLCODE IS sqlcode, DBC=(sqldbcex,sqldbcin),
DSF=(sqldsfcd), PGM=(sqlerrp), QMCD=(q-command)
```

CA-IDMS

```
SQL ERROR, SQLCODE IS sqlcode, SQLCER is sqlcer.
```

SQL Error Message Text

For each SQL error condition reported in the above format, the message associated with the SQLCODE is also returned. The actual message text is obtained from the following:

DB2 for OS/390 and z/OS

The message returned is obtained from the DB2 for OS/390 and z/OS module DSNTIAR.

DB2 for VSE

The message is obtained from the CA-Pan/SQL module DQSMMTB. This module extracts messages from the DB2 for VSE help tables as part of the CA-Pan/SQL installation process.

ORACLE

The message is obtained from an internal ORACLE facility.

CA-Datacom

The message is obtained from the SQLERRM field of the SQLCA.

CA-IDMS

The message is obtained from the SQLCERM field of the SQLCA. For SQL syntax errors, field SQLCER contains the displacement into the SQL statement where the error was found.

Chapter 3

Library Section Definition

Before SQL data can be accessed, you must define the fields to hold the columns to retrieve. These fields are known as host variables.

If you are using native SQL commands or using automatic retrieval without a file, you usually define the fields as working storage fields. Alternatively, you can define the fields in an active output file. This is an effective method to select SQL data into a sequential file for extraction purposes. You must specify which columns to retrieve and which host variables are to receive the data.

When using a CA-Easytrieve Plus file, however, fields defined in the file correspond to the selected columns of the SQL table. The table columns are retrieved into the file fields.

SQL Catalog INCLUDE Facility

You can use the SQL catalog INCLUDE facility to automatically generate CA-Easytrieve Plus field definitions directly from the SQL catalog. This eliminates the need to code host variable definitions in the library section of your program.

The SQL INCLUDE statement names the SQL table or view from which column names and data types are obtained, and defines the location at which the field definitions are generated. The SQL INCLUDE statement must precede any other SQL or SELECT statements and must be coded in the library section of your CA-Easytrieve Plus program.

Note: ORACLE does not provide a catalog INCLUDE facility.

SQL INCLUDE Statement

The CA-Easytrieve SQL INCLUDE statement indicates that SQL table information generates CA-Easytrieve field definitions. It names the table and gives the location where the field definitions are generated.

Note: The SQL INCLUDE statement cannot be used with batch to automatically generate definitions with UPDATE.

Syntax

```
SQL INCLUDE +

 [(column ...)] +

[          {starting-position} ]
[          {* [+offset]      } ]
[ LOCATION {                 } ] +
[          {W                } ]
[          {S                } ]

[HEADING] +

[UPDATE] +

[NULLABLE] +

FROM [owner.] table
```

Parameters

[(column ...)]

Specify a list of one or more column names for which field definitions are generated. The column names must be enclosed in parentheses. If no column names are specified, all columns from the table are used.

```
[          {starting-position} ]
[          {* [+offset]      } ]
[ LOCATION {                 } ]
[          {W                } ]
[          {S                } ]
```

Use this optional parameter to specify the location at which the field definitions are generated.

Starting-position specifies the starting position relative to position one of the record or file.

The * (asterisk) indicates that the field begins in the next available starting position (highest position assigned so far, plus 1). The optional +*offset* is an offset you want added to the * value. There must be at least one blank between the * and the optional +*offset.*

Coding W or S establishes a working storage field. W fields are spooled to report (work) files, S fields are not. W is the default location if the LOCATION parameter is not coded.

`[HEADING]`

Optionally, code HEADING to copy *LABEL* from the DBMS system catalog column entry into a HEADING parameter on the generated DEFINE statement for the column.

`[UPDATE]`

Code UPDATE to designate a modifiable column.

When a CA-Easytrieve SQL file does not contain the UPDATE parameter, only the specific columns defined with UPDATE can be modified with an UPDATE statement. If UPDATE is coded on the FILE statement, you can modify all columns in the file, provided you have the proper authorization.

Note: You can only use UPDATE when the field definitions are generated for a CA-Easytrieve file.

`[NULLABLE]`

Optionally, code NULLABLE to define default indicator fields for columns that contain NULL. The indicator field is defined as a 2 B 0 field preceding the field being defined. CA-Easytrieve automatically uses the default null indicator whenever the associated column is referenced. You can override the use of the default null indicator by explicitly coding and referencing another indicator variable.

The indicator variable precedes the data portion of the field in storage. This field cannot be directly referenced. To check this indicator variable, you must use the IF NULL statement.

`FROM [owner.] table`

FROM identifies the table definition to define to CA-Easytrieve. Owner is the optional 1 to 18-character alphanumeric qualifier, and table is the 1 to 32-character alphanumeric name. You must use the period as the qualification separator for owner-qualified tables.

Note: If the owner is not specified, the current authorization ID is used.

Notes

The generated CA-Easytrieve field names are the same as the SQL column names. If a name matches a reserved word, the field definition is allowed, but all references to it must be qualified using any applicable qualification.

Mask information is not retrieved from the DBMS system catalog.

Group qualification structures of owner.table are defined before the first INCLUDEd definition. The fields are defined under the table entity, which is, in turn, under the owner-level entity. This ensures that multiple tables with duplicate column names do not produce duplicate field names.

Fields with SQL data types that do not have equivalent CA-Easytrieve data types are defined as shown in the following table. You cannot use fields of DATE, TIME, TIMESTAMP, and BINARY in arithmetic operations. Fields of FLOAT, DOUBLEPRECISION, REAL, and LONGINTEGER are defined as packed decimal fields. Non-zero FILE-STATUS and SQLCODE values are returned if the data is truncated.

SQL Data Type	CA-Easytrieve Data Type	Length	Decimals
DATE	Alphanumeric	10	
TIME	Alphanumeric	8	
TIMESTAMP	Alphanumeric	26	
BINARY	Alphanumeric	Length of SQL field	
FLOAT	Packed Numeric	10	3
DOUBLEPRECISION	Packed Numeric	10	3
REAL	Packed Numeric	10	3
LONGINTEGER	Packed Numeric	10	0

The DBMS system catalog must be referenced each time the program is compiled or interpreted. Therefore, to reduce catalog contention and to improve performance, you should always create link-edited programs.

Field Reference

One of the advantages of using the SQL INCLUDE interface is the ability to reference host-variable (CA-Easytrieve fields) using the group level TABLE definition.

When specifying the *INTO* clause on a native *SQL FETCH* or non-file *SQL SELECT* statement or the *VALUES* clause of the native *SQL INSERT* statement, you can substitute the host variable *TABLE* definition in place of coding all host-variables in the table.

If you require access to an indicator variable other than its use for NULL checking, you must define your own variable and reference it with its host-variable. For some DBMSs, the indicator variable is examined to detect truncation.

If NULLABLE is not coded on the INCLUDE statement, an indicator variable can be passed along with the host variable for NULLABLE columns. When the host-variable is a CA-Easytrieve group level definition of a table name, specify an array of type 2 B 0 immediately following the host-table-name-variable. The number of array elements should match the number of fields in the CA-Easytrieve table name definition. Array elements are matched one-to-one with the fields defined in the table name.

Qualifying SQL Column Names

The following example shows how to qualify SQL fields and how to use the keyword NULLABLE.

Table PANSQL.PERSNL3 is defined exactly the same as table PANSQL.PERSNL. To reference a column of table PANSQL.PERSNL, you must qualify it with the table name PERSNL followed by a colon (:).

When you code NULLABLE on an SQL INCLUDE statement, CA-Easytrieve Plus generates the two-byte binary field in front of the column field in the file or record layout. When you code NULLABLE, you can then test the field for being null rather than testing a null indicator.

```
PARM   DEBUG(PMAP DMAP)  ABEXIT  NO
*****************************************************************
***  TABLE PANSQL.PERSNL CONTAINS NULLABLE COLUMNS
*****************************************************************
 FILE SQLFILE SQL
  SQL INCLUDE LOCATION W NULLABLE FROM PANSQL.PERSNL
*****************************************************************
***  TABLE PANSQL.PERSNL3 IS IDENTICAL TO PANSQL.PERSNL
***  AND IS INCLUDED FOR THE PURPOSE OF DEFINING SAME NAMED FIELDS
*****************************************************************
 SQL INCLUDE LOCATION W FROM PANSQL.PERSNL3
 JOB INPUT SQLFILE
           SELECT * FROM PANSQL.PERSNL                         +
          INTO :PANSQL.PERSNL
 IF PERSNL:EMP_PAY_NET NULL
    PERSNL:EMP_PAY_NET = 0
 END-IF
 IF PERSNL:EMP_PAY_GROSS NULL
    PERSNL:EMP_PAY_GROSS = 0
 END-IF
 IF SQLCODE NE 0 AND SQLCODE NE +100
   DISPLAY ‘SQLCODE = ‘ SQLCODE
   STOP EXECUTE
 END-IF
PRINT RPT
REPORT RPT
SEQUENCE PERSNL:EMP_REGIONS
LINE PERSNL:EMP_SSN PERSNL:EMP_NBR PERSNL:EMP_PHONE_NBR
```

Processing NULLable Fields

CA-Easytrieve Plus supports the SQL concept of a null data value.

Null is a value that denotes the absence of a known value for a field. Specify the keyword NULLABLE on the SQL INCLUDE statement to generate the null indicator variables. The rest of the processing is done for you when processing the SQL table as a file.

When a field is defined as nullable, you can use special processing statements:

- You can use IF NULL to determine if the field contains a null value.
- You can use MOVE NULL to set a field's value to null.

Manual NULL Processing

When you use native SQL statements or automatic retrieval without a file, you define null values differently.

You define an indicator variable as a two-byte quantitative binary field (2 B 0). This indicator variable is then used in the INTO clause of the native or automatic SELECT statement. SQL returns a negative value to the indicator variable when the field's value is null. See the native SQL examples in the "Examples" chapter for the use of manual indicator values.

SQL Data Types

The following table illustrates SQL data types and corresponding CA-Easytrieve Plus field definitions. SQL provides some data conversion in SQL assignments and comparisons. Refer to your SQL guides for more information on SQL data conversions.

SQL	CA-Easytrieve Plus	DB2	CA-Datacom/ DB SQL	ORACLE	CA-IDMS SQL
INTEGER	4 B 0 4 I 0	Y	Y	Y	Y
SMALL INTEGER	2 B 0 2 I 0	Y	Y	Y	Y
DECIMAL (x,y)	x P y	Y	Y	Y	Y
UNSIGNED DECIMAL (x,y)	x P y	N	N	N	Y
CHARACTER (x)	x A	Y	Y	Y	Y
VARCHAR (x)	x A VARYING (x <= 254)	Y	Y (8.1)	Y	Y
LONG VARCHAR (x)	x A VARYING (x > 254)	Y	Y (8.1)	Y	Y
NUMERIC (x,y)	x N y	N	Y	Y	Y
UNSIGNED NUMERIC (x,y)	x N y	N	N	N	Y
FLOAT	10 P 3	Y	Y	Y	Y
REAL	10 P 3	Y	Y	N	Y
DOUBLE PRECISION	10 P 3	Y	Y	N	Y
GRAPHIC (x)	x M x K	Y	N	N	Y

SQL	CA-Easytrieve Plus	DB2	CA-Datacom/ DB SQL	ORACLE	CA-IDMS SQL
VARGRAPHIC (x)	x M VARYING x K VARYING (x <= 254)	Y	N	N	Y
LONG VARGRAPHIC (x)	x M VARYING x K VARYING (x > 254)	Y	N	N	Y
DATE	10 A	Y	Y	N	Y
DATE (ORACLE)	12 A	N	N	Y	N
RAW	x A	N	N	Y	N
LONGRAW	x A VARYING	N	N	Y	N
TIME	8 A	Y	Y	N	Y
TIMESTAMP	26 A	Y	Y	N	Y
LONG INTEGER	10 P 0	N	N	N	Y
BINARY	x A y	N	N	N	Y
none	x U y	-	-	-	-

Decimal Data Types

For SQL DECIMAL data types, the scale is the same as the decimal places of a CA-Easytrieve Plus field. SQL precision refers to the total number of digits that can occur in the packed field.

Length refers to the number of bytes occupied by the packed field.

A field that is 5 P 2 is the equivalent of an SQL DECIMAL data type of precision = 9 and scale = 2. Depending on your SQL release, SQL might not support CA-Easytrieve Plus packed fields with lengths > 8.

To ensure even precision, you should specify the keyword EVEN on the DEFINE for user-coded host variables.

SQL Syntax Checking

When an SQL statement is passed to SQL for syntax checking, host variables are converted to question marks (?). It is possible that when an SQL error is detected, the question mark is identified as the field in error. In this case, you are responsible for looking up the error message and identifying which host variable is in error.

Because host variables are replaced with question marks, their use in arithmetic expressions can result in compile errors. For DB2 for OS/390 and z/OS and DB2 for VSE, an SQLCODE of -418 can occur.

System-Defined File Fields

When using a CA-Easytrieve Plus file to process an SQL database, two system-defined fields are used:

- RECORD-COUNT
- RECORD-LENGTH.

RECORD-COUNT

RECORD-COUNT contains the number of rows returned to the CA-Easytrieve Plus program. This is the number of rows fetched either by automatic or controlled processing.

RECORD-LENGTH

RECORD-LENGTH is the length of the SQL file. The length is the sum of the maximum lengths of all fields in the file.

SQL Communications Area Fields

All of the SQL Communication Area fields (SQLCA) are automatically created in **S** (static) working storage when any of the following occurs:

- The first SQL-managed FILE IS ENCOUNTERED
- The first SQL INCLUDE statement is encountered
- The first native SQL statement is found
- The first JOB INPUT SQL statement is found.

The fields, generated for SQLCA for the supported SQL database management systems, are shown below.

```
DEFINE SQLCA          S           136  A
DEFINE SQLCAID        SQLCA         8  A
DEFINE SQLCABC        SQLCA +8      4  B  0
DEFINE SQLCODE        SQLCA +12     4  B  0
DEFINE SQLERRM        SQLCA +16    72  A
DEFINE SQLERRML       SQLCA +16     2  B  0
DEFINE SQLERRMC       SQLCA +18    70  A
DEFINE SQLERRP        SQLCA +88     8  A
DEFINE SQLERRD        SQLCA +96     4  B  0 OCCURS 6
DEFINE SQLWARN        SQLCA +120    8  A
DEFINE SQLWARN0       SQLCA +120    1  A
DEFINE SQLWARN1       SQLCA +121    1  A
DEFINE SQLWARN2       SQLCA +122    1  A
DEFINE SQLWARN3       SQLCA +123    1  A
DEFINE SQLWARN4       SQLCA +124    1  A
DEFINE SQLWARN5       SQLCA +125    1  A
DEFINE SQLWARN6       SQLCA +126    1  A
DEFINE SQLWARN7       SQLCA +127    1  A
DEFINE SQLEXT         SQLCA +128    8  A
DEFINE SQLWARN8       SQLCA +128    1  A
DEFINE SQLWARN9       SQLCA +129    1  A
DEFINE SQLWARNA       SQLCA +130    1  A
DEFINE SQLSTATE       SQLCA +131    5  A


SQLCA                 S           196  A
SQLCA-EYE-CATCH       SQLCA         8  A
SQLCAID               SQLCA         8  A
SQLCA-LEN             SQLCA +8      4  B  0
SQLCABC               SQLCA +8      4  B  0
SQLCA-DB-VRS          SQLCA +12     2  A
SQLCA-DB-RLS          SQLCA +14     2  A
SQLCA-LUWID           SQLCA +16     8  A
SQLCODE               SQLCA +24     4  B  0
SQLCA-ERROR-INFO      SQLCA +28    82  A
SQLCA-ERR-LEN         SQLCA +28     2  B  0
SQLCA-ERR-MSG         SQLCA +30    80  A
SQLERRM               SQLCA +28    72  A
SQLERRML              SQLCA +28     2  B  0
SQLERRMC              SQLCA +30    70  A
SQLCA-ERROR-PGM       SQLCA +110    8  A
SQLERRP               SQLCA +110    8  A
SQLCA-FILLER-1        SQLCA +118    2  A
SQLCA-ERROR-DATA      SQLCA +120   24  A
SQLCA-DSFCODE         SQLCA +120    4  A
SQLCA-INFCODE         SQLCA +124    4  B  0
SQLCA-DBCODE          SQLCA +128    4  A
SQLCA-DBCODE-EXT      SQLCA +128    2  A
SQLCA-DBCODE-INT      SQLCA +130    2  B  0
SQLCA-MISC-CODE1      SQLCA +132    4  A
SQLCA-MISC-CODE2      SQLCA +136    4  B  0
SQLCA-MISC-CODE3      SQLCA +140    4  A
SQLCA-WRN-AREA        SQLCA +144    8  A
SQLCA-WARNING         SQLCA +144    1  A           OCCURS 8
SQLWARN               SQLCA +144    8  A
SQLWARN0              SQLCA +144    1  A
SQLWARN1              SQLCA +145    1  A
SQLWARN2              SQLCA +146    1  A
SQLWARN3              SQLCA +147    1  A
```

```
SQLWARN4              SQLCA +148      1  A
SQLWARN5              SQLCA +149      1  A
SQLWARN6              SQLCA +150      1  A
SQLWARN7              SQLCA +151      1  A
SQLCA-PGM-NAME        SQLCA +152      8  A
SQLCA-AUTHID          SQLCA +160     18  A
SQLCA-PLAN-NAME       SQLCA +178     18  A



SQLCA                 S               344  A
SQLCAID               SQLCA           8    A
SQLCODE               SQLCA +8        4    B O
SQLCSID               SQLCA +12       4    B O  OCCURS 2
SQLCERC               SQLCA +20       4    B O
SQLCNRP               SQLCA +28       4    B O
SQLCSER               SQLCA +36       4    B O
SQLCLNO               SQLCA +44       4    B O
SQLCMCT               SQLCA +48       4    B O
SQLCOPTS              SQLCA +52       4    B O
SQLCFJB               SQLCA +56       4    B O
SQLCPCID              SQLCA +60       4    B O
SQLCLCID              SQLCA +64       4    B O
SQLCERL               SQLCA +68       2    B O
SQLCERM               SQLCA +72       256  A
SQLSTATE              SQLCA +328      5    A

DEFINE SQLCA          S             136  A
DEFINE SQLCAID        SQLCA           8  A
DEFINE SQLCABC        SQLCA +8        4  B  0
DEFINE SQLCODE        SQLCA +12       4  B  0
DEFINE SQLERRM        SQLCA +16      72  A
DEFINE SQLERRML       SQLCA +16       2  B  0
DEFINE SQLERRMC       SQLCA +18      70  A
DEFINE SQLERRP        SQLCA +88       8  A
DEFINE SQLERRD        SQLCA +96       4  B  0 OCCURS 6
DEFINE SQLWARN        SQLCA +120      8  A
DEFINE SQLWARN0       SQLCA +120      1  A
DEFINE SQLWARN1       SQLCA +121      1  A
DEFINE SQLWARN2       SQLCA +122      1  A
DEFINE SQLWARN3       SQLCA +123      1  A
DEFINE SQLWARN4       SQLCA +124      1  A
DEFINE SQLWARN5       SQLCA +125      1  A
DEFINE SQLWARN6       SQLCA +126      1  A
DEFINE SQLWARN7       SQLCA +127      1  A
DEFINE SQLWARN8       SQLCA +128      1  A
DEFINE SQLWARN9       SQLCA +129      1  A
DEFINE SQLWARNA       SQLCA +130      1  A
DEFINE SQLEXT         SQLCA +131      5  A
```

Sample Database

The following example illustrates the two tables used for all the examples in this chapter:

```
TABLE:    PERSONNEL
          ----------------------------------
COLUMNS:  EMPNAME                WORKDEPT   EMPPHONE   SALARY
          ----------------------------------
DATA:     NORIDGE DEBBIE         901         5001       32400
          OSMON   SAMUEL         901         5004       62800
          MILLER  JOAN           950         6034       31360
          EPERT   LINDA          950         null       31040
          STRIDE  ANN            901         null       38640
          ROGERS  PAT            921         2231       32900


EMPNAME       -  CHAR(20) (NOT NULL)
WORKDEPT      -  DECIMAL(3,0) (NOT NULL)
EMPPHONE      -  DECIMAL(5,0) (NULL)
SALARY        -  DECIMAL(8,2) (NOT NULL)
```

```
TABLE:    DEPARTMENTS
          ------------------------------
COLUMNS:  DEPTNAME            DEPTNUMBER
          ------------------------------
DATA:     SHIPPING            901
          HUMAN RESOURCES     921
          ACCOUNTING          950
          DATA PROCESSING     951

DEPTNAME      -   VARCHAR(20)  (NOT NULL)
DEPTNUMBER    -   DECIMAL(3,0)  (NOT NULL)

WORKDEPT in the PERSONNEL table corresponds with
the DEPTNUMBER in the DEPARTMENTS table.
```

Working Storage Definitions

The following example shows working storage field definitions for the sample tables in the previous example:

```
DEFINE EMPNAME    W 20 A
DEFINE WORKDEPT   W  2 P 0
DEFINE EMPPHONE   W  3 P 0
DEFINE DEPTNAME   W 22 A    VARYING
DEFINE DEPTNUMBER W  2 P 0
DEFINE NULLPHONE  W  2 B 0 .* NULL INDICATOR
DEFINE SYS-USERID W  8 A   VALUE('SQLDBA')     .* SQL/DS USERID
DEFINE PASSWORD   W  8 A   VALUE('SQLDBAPW')   .* SQL/DS PASSWORD
```

Chapter 4

Automatic Processing

This chapter describes automatic table processing. With automatic processing, you can retrieve selected data (or all data) from every row in a table or view.

Files Associated with SQL Cursor

SQL cursor management can be automated when you associate an SQL cursor with a CA-Easytrieve Plus file. The SQL file can then be accessed with the JOB INPUT statement. With each iteration of the JOB statement or activity, another row from the table is automatically retrieved into the file's data area. Even if you only have a basic knowledge of SQL, you can report on data contained in an SQL database.

Automatic Retrieval Without a File

Automatic retrieval does not require that you define a CA-Easytrieve Plus file. In this read-only method, SQL must be coded on the JOB statement in place of a file name. You must code a SELECT statement directly after the JOB statement to specify the columns to retrieve and the host variables to receive the data. Each time the JOB activity is iterated, another row of SQL data is retrieved. This is a simple way to retrieve SQL data into working storage or into an extract file for subsequent output.

Specifying Automatic Input

Automatic input is specified using the JOB statement and either the SELECT statement with the select-clause, the FILE statement with the SQL keyword plus select-clause, or a combination of these statements. The possible approaches are as follows:

1. JOB INPUT SQL
   SELECT select-clause

   The JOB statement uses the SQL keyword to indicate that input is coming from an SQL table. The SELECT statement specifies the select-clause, which identifies the table and provides other parameters for automatic input.

2. FILE filename SQL (select-clause)
   JOB INPUT filename

   The FILE statement uses the SQL keyword to indicate that filename is an SQL file and uses the select-clause to specify parameters for automatic input. The JOB statement names the SQL file.

3. FILE filename SQL
   JOB INPUT filename
   SELECT select-clause

   The FILE statement uses the SQL keyword to indicate that filename is an SQL file. The JOB statement names the SQL file. The SELECT statement identifies the information to retrieve. The SELECT statement is required since no select-clause was coded on the FILE statement.

4. FILE filename SQL (select-clause)
   JOB INPUT filename
   SELECT select-clause

   This is the same as approach 2 above, except that the select-clause on the SELECT statement overrides the select-clause on the FILE statement.

Note: If the select-clause is not coded on the FILE statement, the SELECT statement must immediately follow the JOB statement.

FILE Statement

The FILE statement can identify a file as an SQL file and specify the select-clause. When used in this way, the FILE statement is limited to the following syntax.

```
FILE filename  SQL [(select-clause)]      +
               [DEFER]                    +

               [         {IBM     } ]
               [         {IBMKOREA} ]
               [         {JEF     } ]
               [         {JEF4040 } ]
               [DBCSCODE {JIPSE   } ]
               [         {JIS     } ]
               [         {KEIS    } ]
               [         {MELCOM  } ]
               [         {SHOWA   } ]
               [         {TORAY   } ]
```

`filename`

This is the SQL filename. This filename can subsequently be referenced on the JOB statement or on a GET statement. Filename can be used as a synchronized file.

```
SQL [(select-clause)]
```

SQL identifies filename as an SQL file. Select-clause is specified in the same manner as the select-clause used on the SELECT statement (see Select-Clause Syntax and Select-Clause Syntax in the description of the SELECT statement).

- If select-clause is coded, it must be in parentheses.
- If select-clause is not coded, a SELECT statement containing the select-clause must be coded following the JOB statement.
- If select-clause is not coded, you cannot use filename on a GET statement.

```
[DEFER]
```

DEFER opens an SQL cursor at the execution of the first GET statement. This means that the cursor is opened after the user-specified start procedure. If DEFER is not coded, the cursor is opened before the user-specified start procedure.

```
[DBCSCODE]
```

DBCSCODE defines the code system that is associated with all CA-Easytrieve Plus fields defined for this file.

- If you do not code this option, the Processing code system is used. The DBCS Options module identifies this code system. You can alter the code system by using the DBCSCODE option of the PARM statement.
- If your site does not support the DBCS option, then this option is invalid.
- If the file being defined is associated with an extended reporting printer that does not support DBCS data, then this keyword is invalid. Should the extended reporting printer support DBCS data, then you can use the DBCSCODE keyword to modify the DBCS code system of the printer.

Refer to the CA-Easytrieve Plus *Getting Started* for more information.

JOB Statement

The syntax of the JOB statement for SQL is as follows.

```
           {filename}
JOB INPUT  {SQL     }    . . .
           {        }
```

When an SQL filename is specified on a JOB statement, input is based on the select-clause given on the FILE statement. If the keyword SQL is coded on the JOB statement, input is based on the select-clause given on the SELECT statement immediately following the JOB statement. Any nonzero SQLCODE condition from the select-clause terminates execution.

Coding SQL or an SQL filename on the JOB statement opens an SQL cursor at the start of the job activity. An SQL FETCH command or statement executes for each execution of the JOB statement (including the first).

SELECT Statement

Code the SELECT statement immediately following the:

- JOB INPUT SQL statement, in which case the SELECT statement is required.
- JOB INPUT filename statement where:
  - The select-clause was not coded on the FILE statement. In this case, the SELECT statement is required.
  - You want to override the select-clause that was given on the FILE statement. In this case, the SELECT statement is optional.

The SELECT statement, or the select-clause on the FILE statement if coded, identifies the rows and columns that are to be input to the JOB activity. You can code only one SELECT statement in each JOB activity.

Select-Clause Syntax

The select-clause identifies the rows and columns that are input to the JOB activity.

```
          [        ]  {{   *               }
          [DISTINCT]  {{expression         }
  SELECT  [  ALL   ]  {{table-name.*       }      +
          [        ]  {{correlation-name. *}

               [  {expression         }     ]  }
               [, {table-name.*       } ... ]  }  +
               [  {correlation-name.*  }     ]  }
               [  {                   }     ]  }

  FROM table-name [correlation-name] +
       [,table-name [correlation-name] ... ]     +

 [WHERE search-condition]                      +

   [                              ]
   [GROUP BY column-name        + ]              +
   [        [, column-name ...]   ]
   [                              ]

   [HAVING search-condition]                     +
[                                                         ]
[                [        ]  {{   *              }        ]
[                [DISTINCT]  {{expression        }        ]
[UNION   SELECT  [  ALL   ]  {{table-name.*      } +      ]
[                [        ]  {{correlation-name.*}        ]
[                                                         ]
[                [  {expression       }     ]  }          ]
[                [, {table-name.*     } ... ]  }  +       ]
[                [  {correlation-name.*}    ]  }          ]
[                [  {                 }     ]  }          ]
[                                                         ]
[  FROM table-name [correlation-name] +                   ]
[        [,table-name [correlation-name] ... ]     +      ] +
[                                                         ]
[    [WHERE search-condition]                      +      ]
[                                                         ]
[    [                              ]                     ]
[    [GROUP BY column-name        + ]              +      ]
[    [        [, column-name ...]   ]                     ]
[    [                              ]                     ]
[                                                         ]
[    [HAVING search-condition]                            ]
[                                                         ]

[         {           }  [    ]           ]
[ORDER BY {column-name}  [ASC ]         + ]
[         { integer   }  [DESC]           ]
[         {           }  [    ]           ]
[                                         ]
[         [  {           } [    ]     ]     ]
[         [, {column-name} [ASC ]     ]     ]
[         [  {integer    } [DESC] ... ]     ]          +
[         [  {           } [    ]     ]     ]

 INTO :host-variable [, :host-variable...]
```

```
[        ]
[DISTINCT]
[ALL     ]
[        ]
```

DISTINCT eliminates duplicate rows. ALL specifies that duplicate rows are not eliminated. ALL is the default.

```
{   *              }
{expression        }
{table-name.*      }
{correlation-name.*}
```

These identify the columns to retrieve from the specified table.

```
FROM table-name [correlation-name]
```

Table-name specifies the table from which data is retrieved. Correlation-name can define an alias for the table-name that immediately precedes the correlation-name.

```
[WHERE search-condition]
```

Search-condition specifies conditions for the retrieval of data. The search-condition is applied to the result of the FROM clause. Refer to your SQL guides for a description of the search-condition.

```
[GROUP BY column-name]
```

GROUP BY groups data from the FROM and WHERE clauses. Column-name cannot be a long string (more than 254 bytes in length).

```
[HAVING search-condition]
```

Search-condition also specifies the data provided to the user. HAVING can compare the results of all the returned data with a specific value in the data provided, such as the minimum or maximum value. Refer to your SQL guides for a description of the search-condition.

```
[UNION...]
```

The UNION verb includes rows from another table.

```
[         {           }  [     ] ]
[ORDER BY {column-name}  [ASC ] ]
[         {integer    }  [DESC] ]
[         {           }  [     ] ]
```

ORDER BY returns the rows of the result table in the order of the values of the specified column-names. ASC returns the rows in ascending order and is the default. DESC returns the rows in descending order. Integer references a column by its position rather than by a column-name.

```
INTO :host-variable
```

The INTO clause identifies where the column values are placed. The INTO clause must be the last clause coded in the select-clause.

Select-Clause Operation

If this execution is for an DB2 for VSE or ORACLE system, a CONNECT statement is generated and executed. This means that the user does not need to include an SQL CONNECT statement when using CA-Easytrieve Plus automatic processing. The user ID and password values are those that were specified in the USERID parameter of the PARM statement.

The SQLCODE field is checked following each execution of the select-clause.

- If the SQLCODE is a value other than a zero (0) or a no more rows found condition, an error message is issued based on the SQL error and execution terminates.
- A no more rows found condition causes the initiation of end of input processing: the FINISH PROC (if any) executes, spooled reports are printed, and the current JOB activity ends.

The select-clause on a FILE statement permits references to fields not yet defined, but all references must be defined by the first JOB or SORT statement.

The SQL cursor that is automatically defined by a SELECT statement or a FILE statement (with the SQL option) is closed following the JOB activity referencing it.

For FILE statements with the SQL option:

- When the select-clause encounters no more rows found condition, the SQL file is marked EOF (end-of-file). In the case where the JOB statement references the SQL file, execution of the JOB activity stops and the FINISH procedure (if present) executes.
- The system-defined field RECORD-COUNT contains the number of rows returned.
- The length of the SQL file is the sum of the maximum lengths of all the fields in the file. The system-defined field RECORD-LENGTH contains this value.
- Fields are defined to the SQL file using any available means.
- The select-clause cannot reference a CA-Easytrieve Plus group level definition of a table name as a host variable.
- File statistics are produced at the end of execution of each JOB activity.

Generated Code

The pseudo-code, generated for automatic SQL processing, is:

```
* IF SQL/DS, ORACLE, OR CA-IDMS
     SQL CONNECT  . . .
* END-IF
  SQL DECLARE cursor CURSOR FOR select clause
  SQL OPEN cursor
  DO WHILE SQLCODE NE 100
     SQL FETCH cursor INTO :host-variable  +
                        [, :host-variable...]
     process EZT+ code
  END-DO
  SQL CLOSE cursor
```

Chapter 5

Controlled Processing

Controlled SQL Processing can be performed using the Native SQL commands or the GET statement if SQL files are used.

GET Statement

The syntax for the GET statement for SQL is as follows.

Syntax

```
GET filename  [STATUS]
```

When an SQL filename is specified on a GET statement, an SQL FETCH statement is issued, and the next available row is returned from the SQL cursor. The STATUS keyword returns the SQLCODE in the system-defined field FILE-STATUS (defined as 4 B 0). If STATUS is not coded, FILE-STATUS contains zeros. To use an SQL filename on GET, code the select-clause on the FILE statement.

An SQL filename on the GET statement opens an SQL cursor at the start of the job activity, unless DEFER is coded on the FILE statement. If DEFER is coded, an SQL cursor opens at the execution of the first GET statement.

If the STATUS keyword is not coded on the GET statement for an SQL file, a nonzero SQLCODE condition terminates execution.

Remember, for FILE statements with the SQL options:

- When the select-clause encounters an SQL no more rows condition, the SQL file is marked EOF.
- The system-defined field RECORD-COUNT contains the number of rows returned.
- The length of the SQL file is the sum of the maximum lengths of all the fields in the file. The system-defined field RECORD-LENGTH contains this value.
- Fields are defined to the SQL file using any available means.

- The select-clause cannot reference a CA-Easytrieve Plus group-level definition of a table name as a host variable.
- File statistics are produced at the end of each JOB activity execution.

Native SQL

This method of processing uses native SQL statements that are equivalent to many of those used in COBOL. By using these native SQL statements, you control the SQL cursor operation. All native SQL statements are prefixed with the SQL keyword.

Syntax

```
SQL native-sql-statement
```

Usage Notes

See the specific database management system guide for information about syntax for native database statements. Listed below are the SQL statements currently supported by the SQL interface.

DB2 for OS/390 and z/OS SQL Statements:

- ALLOCATE cursor-name *(for static-only processing)
- ALTER
- ASSOCIATE *(for static-only processing)
- CALL procedure-name *(for static-only processing)
- CLOSE cursor-name
- COMMENT ON
- COMMIT {work}
- CONNECT
- CREATE
- DECLARE cursor-name {with hold}
- DELETE {where current of cursor-name}
- DROP
- EXPLAIN
- FETCH cursor-name
- GRANT

- INSERT
- LABEL
- LOCK
- OPEN cursor-name
- RELEASE
- REVOKE
- ROLLBACK {work}
- SELECT INTO *(for static-only processing)
- SET CONNECTION
- SET CURRENT DEGREE
- SET CURRENT PACKAGESET
- SET CURRENT SQLID
- SET host-variable
- UPDATE {where current of cursor-name}.

* Refer to SQLSYNTAX in the PARM statement.

DB2 for VSE SQL Statements:

- ACQUIRE
- ALTER
- CLOSE cursor-name
- COMMENT
- COMMIT {work}
- CONNECT :userid
- CONNECT TO :database
- CREATE
- DECLARE CURSOR-NAME
- DELETE {where current of cursor-name}
- DROP
- EXPLAIN
- FETCH cursor-name
- GRANT
- INSERT
- LABEL

- LOCK
- OPEN cursor-name
- PUT
- REVOKE
- ROLLBACK {work}
- UPDATE {where current of cursor-name}.

CA-Datacom/DB SQL Statements:

- ALTER
- CLOSE cursor-name
- COMMENT
- COMMIT {work}
- CREATE
- DECLARE cursor-name
- DELETE {where current of cursor-name}
- DROP
- FETCH cursor-name
- GRANT
- INSERT
- LOCK
- OPEN cursor-name
- REVOKE
- ROLLBACK {work}
- SELECT INTO
- UPDATE {where current of cursor-name}.

CA-IDMS SQL Statements:

- ALTER
- CLOSE cursor-name
- COMMIT {work} {continue} {release}
- CONNECT TO dictionary-name
- CREATE
- DECLARE cursor-name
- DELETE where search-condition *

- DROP
- EXPLAIN
- FETCH cursor-name
- GRANT
- INSERT
- OPEN cursor-name
- RELEASE
- RESUME
- REVOKE
- ROLLBACK {work}
- SUSPEND
- UPDATE where search-condition *.
- The DELETE and UPDATE can only execute using a search condition on the wher*e* clause. The “where current of cursor-name” syntax is not supported using dynamic SQL processing.

ORACLE Statements:

- DECLARE
- OPEN
- FETCH
- CLOSE
- UPDATE
- INSERT
- DELETE
- CONNECT (by parms or identified by password)
- COMMIT
- ROLLBACK
- ALTER
- COMMENT
- CREATE
- DROP
- GRANT
- LOCK
- REVOKE

- AUDIT
- NOAUDIT
- RENAME
- VALIDATE SET

Processing Requirements

- The SQL DECLARE statement must be coded in the Library Definition section of a CA-Easytrieve Plus program. All other SQL statements, except SQL INCLUDE, must be coded in the Activity Definition section.
- You should test the SQLCODE field in the SQLCA to determine if the execution of each controlled processing statement is successful.

  If the SQLCODE field contains a zero (0), you should test the SQLWARN0 field to ensure that no warning conditions were issued during processing of the SQL statement. Refer to the appropriate SQL reference guide to determine acceptable values for SQLWARN0.
- All SQL INCLUDE statements and SQL-managed file definitions must be coded before any controlled SQL statements.

Operation

Coding native SQL statements requires an advanced knowledge of SQL statements and of the database to process. Native SQL statements can be coded in any JOB activity. You cannot code them in SORT or REPORT procedures.

CALL Statement

The syntax for the CALL statement as supported by DB2 for OS/390 and z/OS is as follows.

CALL Syntax: Format 1

```
CALL procedure-name ( parmlist )
```

where the procedure name is provided as a host variable

CALL Syntax: Format 2

```
CALL :procedure-name ( parmlist )
```

where the procedure name is provided as a host variable

Format A is the only supported format fo the CALL Statement.

Format B is **not** supported. CA-Easytrieve Plus performs some syntax checking of SQL statements, replacing host variable identifiers with a question mark. In the case of Foramt B, it identifies the parameter list as a subscript of the variable procedure name and flags this as an error.

CALL, ASSOCIATE, and ALLOCATE

To use the CALL, ASSOCIATE, and ALLOCATE statements, you must have a BIND option of STATIC-ONLY. This requires the JCL shown for the EXTPDB2 procedure. (This JCL is shown in the EXTPDB2 Procedure section of the "Executing Your SQL Program" chapter.

These statements undergo partial syntax checking by the PanSQL interface. They undergo complete syntax checking when the static-command-program is processed by the DB2 for OS/390 and z/OS preprocessor DSNHPC. The output from the DB2 for OS/390 and z/OS preprocessor must be examined for any possible SQL errors.

See the Static SQL Mode section in the "Executing Your SQL Program" chapter of this guide for a detailed explanation of the steps executed to run the program statically.

Chapter

6

Executing Your SQL Program

This chapter describes how to use CA-Easytrieve Plus in different environments.

DB2 for OS/390 and z/OS Execution

Dynamic SQL Mode

The following example illustrates the JCL necessary to execute CA-Easytrieve Plus with DB2 for OS/390 and z/OS, using the Dynamic SQL mode.

OS/390 and z/OS JCL

```
//jobname   JOB  accounting.info,USER=userid
//stepname  EXEC PGM=EZTPA00
//STEPLIB   DD   DISP=SHR,DSN=your.eztp.loadlib
//          DD   DISP=SHR,DSN=your.pansql.loadlib
//          DD   DISP=SHR,DSN=your.db2.sspgm.lib
//SYSPRINT  DD   SYSOUT=A
//SYSOUT    DD   SYSOUT=A
//SORTWK01  DD   UNIT=SYSDA,SPACE=(CYL,1)
//SYSSNAP   DD   SYSOUT=A
//SYSUDUMP  DD   SYSOUT=A
//EZTVFM    DD   UNIT=SYSDA,SPACE=(4096,(100,100))
//SYSIN     DD   *
 ... CA-Easytrieve Plus DB2 source statements ...
/*
```

Note: The value of `sspgm` names your IBM DB2 for OS/390 and z/OS load library, which contains the programs DSNHLI2 and DSNALI.

Static SQL Mode

This topic describes the additional steps that must run to create the Static application plan.

Note: The JCL procedure (EZTPDB2) for building the static application plan is in the SAMPJCL library downloaded during installation.

Sample Job Stream

The procedure for building the Static application plan follows these steps:

1. The Pan/SQL interface called by CA-Easytrieve Plus writes the SQL statements, along with other control information, to a command program generation file (ddname = GENDATA).
2. The GENDATA file is processed by Pan/SQL to generate a valid assembler program for the IBM DB2 for OS/390 and z/OS preprocessor.
3. The DB2 for OS/390 and z/OS precompiler translates the command program assembler source and creates a DBRM (database request module) entry in the DBRMLIB.
4. The command program can then be assembled and link edited to produce the static command program.
5. The DBRM can then be bound into an application plan or package. The sample JCL creates a plan. You can modify this JCL to bind the DBRM into a package. The final planname must match the planname specified for the PLAN parameter.

PAN$SQL DD File

You can code a PAN$SQL DD statement to provide a plan name and DB2 for OS/390 and z/OS subsystem to be used at the time of your execution. The PAN$SQL DD file is processed only when executing statically and if the DB2 for OS/390 and z/OS Call Attach Facility is used to establish a connection.

The PAN$SQL file can also be used to indicate that you want to execute under the TSO Terminal Monitor Program in background mode. If TSO execution is specified, then the plan name and subsystem ID parameters are ignored.

Valid parameters for the PAN$SQL file are the following:

- PLAN—provide the name of the DB2 for OS/390 and z/OS plan to use for execution
- SSID—provide the name of the DB2 for OS/390 and z/OS subsystem to use for the DB2 for OS/390 and z/OS Call Attach Facility for a connection..
- TSO—indicates that you want to execute your CA-Easytrieve Plus program under the TSO Terminal Monitor Program in background mode. You must code the correct JCL.

  The following is sample JCL for the PAN$SQL statement:

  ```
  //PAN$SQL DD *
      PLAN=TESTPLAN,SSID=D510
  ```

The ability to specify a planname for execution enables you to compile and link your CA-Easytrieve Plus DB2 for OS/390 and z/OS application program once. The DBRM can then be bound into any DB2 for OS/390 and z/OS subsystem with any planname.

Execution under TSO resolves the problem of called DB2 for OS/390 and z/OS subroutines that previously had to be linked with DSNALI, the DB2 for OS/390 and z/OS Call Attach Facility module. This restriction required subroutines to be linked one way for other applications.

If your CA-Easytrieve Plus program contains SQL statements and it also calls COBOL (or other language) subroutines, you can now share those subroutines with CA-Easytrieve Plus applications and other applications. You can link your COBOL subroutines once with DSNELI and it can be used by all of your applications..

Sample JCL for Execution Under TSO

```
//EXECEZT     EXEC PGM=IKJEFT01,DYNAMBR=20
//STEPLIB     DD  DISP=SHR,DSN=your.ezt.target.library
//            DD  DISP=SHR,DSN=your.ezt.db2.application.library
//            DD  DISP=SHR,DSN=your.pansql.tso.load.library
//            DD  DISP=SHR,DSN=your.db2.sdsnload.library
//PAN$SQL     DD  *
   TSO                           <=== Indicate TSO execution
//SYSPRINT    DD  SYSOUT=*
//SYSTSPRT    DD  SYSOUT=*
//SYSUDUMP    DD  SYSOUT=*
//SYSOUT      DD  SYSOUT=*
//REPORT      DD  SYSOUT=*
//SYSIN       DD  DUMMY
//SYSTSIN     DD  *
 DSN SYSTEM(D510)
 RUN PROGRAM(eztpgm) PLAN(testplan)
 END
/*
```

EZTPDB2 Procedure

The EZTPDB2 procedure is installed as member DB2BINDP in your SAMPJCL dataset. The EZTPDB2 procedure builds the static application program and plan.

```
//**********************************************************************
//*
//*   PROC TO:  1)  Compile the CA-Easytrieve Plus program
//*             2)  Link the CA-Easytrieve Plus program
//*             3)  Process the Pan/SQL DB2 static command program
//*             4)  Preprocess the static command program by DB2
//*             5)  Assemble the static command program
//*             6)  Link the static command program
//*             7)  Run TSO to BIND and GRANT authorization
//*
//**********************************************************************
//EZTPDB2     PROC  DBRMLIB=,          /*DBRMLIB                    */
//                 DB2LIB=,            /*DB2 LOAD LIB               */
//                 RUNLIB=,            /*DB2 RUNLIB                 */
//                 EZTPLIB=,           /*EZTP LOAD LIB              */
//                 EZTPPGM=,           /*EZTP LINKED PGM LIB        */
//                 PSQLLIB=,           /*PAN/SQL LOAD LIB           */
//                 DBRMNME=,           /*DBRMNAME                   */
//                 STCPGMN=            /*STATIC.CMD.PROGRAM         */
//*
//**********************************************************************
//*
//*            STEP 1:  Compile CA-Easytrieve Plus program
//*
//**********************************************************************
//EZTPLUS    EXEC PGM=EZTPA00
//STEPLIB    DD   DISP=SHR,DSN=&EZTPLIB
//           DD   DISP=SHR,DSN=&PSQLLIB
//           DD   DISP=SHR,DSN=&DB2LIB
//SYSPRINT   DD   SYSOUT=*
//SYSLIN     DD   DISP=(NEW,PASS),DSN=&&SYSLIN,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(3120,(100,50),RLSE)
//GENDATA    DD   DISP=(NEW,PASS),DSN=&&EZTPDB2,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(3120,(100,50),RLSE)
//EZTVFM     DD   UNIT=SYSDA,SPACE=(CYL,(1,5))
//**********************************************************************
//*
//*            STEP 2:  CA-Easytrieve Plus Link
//*
//**********************************************************************
//LKEDEZT   EXEC PGM=IEWL,COND=(0,NE,EZTPLUS)
//SYSPRINT   DD  SYSOUT=*
//SYSLIN     DD  DISP=(OLD,DELETE) DSN=&&SYSLIN
//SYSLMOD    DD  DISP=SHR,DSN=&EZTPPGM
//SYSUT1     DD  UNIT=SYSDA,SPACE=(CYL,(1,5))
```

```
//*********************************************************************
//*
//*            STEP 3:  Generate static command program statements
//*
//*********************************************************************
//GEN       EXEC PGM=DQSCGEN,COND=(0,NE,EZTPLUS)
//STEPLIB    DD  DISP=SHR,DSN=&PSQLLIB
//GENDATA    DD  DISP=(OLD,DELETE),DSN=&&EZTPDB2
//CMDPGM1    DD  DISP=(NEW,PASS),DSN=&&CPGM1,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(TRK,(1,1))
//CMDPGM2    DD  DISP=(NEW,PASS),DSN=&&CPGM2,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(TRK,(2,1))
//CMDPGM3    DD  DISP=(NEW,PASS),DSN=&&CPGM3,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(TRK,(2,1))
//CMDPGM4    DD  DISP=(NEW,PASS),DSN=&&CPGM4,
//               UNIT=SYSDA,
//               DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB),
//               SPACE=(TRK,(2,1))
//SYSPRINT   DD  SYSOUT=*
//ERRREPT    DD  SYSOUT=*
//*********************************************************************
//
//*            STEP 4:  Preprocess the static command program by DB2
//*                          (DB2 Precompile)
//*
//*********************************************************************
//DB2PRE    EXEC PGM=DSNHPC,COND=(0,NE,EZTPLUS),
//               PARM='HOST(ASM),XREF ,SOURCE'
//STEPLIB    DD  DISP=SHR,DSN=&DB2LIB
//DBRMLIB    DD  DISP=SHR,DSN=&DBRMLIB(&DBRMNME)
//SYSIN      DD  DISP=(OLD,DELETE),DSN=&&CPGM1
//           DD  DISP=(OLD,DELETE),DSN=&&CPGM2
//           DD  DISP=(OLD,DELETE),DSN=&&CPGM3
//           DD  DISP=(OLD,DELETE),DSN=&&CPGM4
//SYSCIN     DD  DSN=&&DB2OUT,DISP=(NEW,PASS),
//               UNIT=SYSDA,DCB=BLKSIZE=3120,
//               SPACE=(TRK,(3,3))
//SYSUT1     DD  SPACE=(800,(1000,1000)),UNIT=SYSDA
//SYSUT2     DD  SPACE=(800,(1000,1000)),UNIT=SYSDA
//SYSPRINT   DD  SYSOUT=*
//SYSTERM    DD  SYSOUT=*
//*********************************************************************
//*
//*            STEP 5:  Assemble the static command program
//*
//*********************************************************************
//ASMCMD    EXEC PGM=IEV90,COND=(0,NE,EZTPLUS),
//               PARM='OBJECT,NODECK,RENT,LIST'
//SYSIN      DD  DISP=(OLD,DELETE),DSN=&&DB2OUT
//SYSLIN     DD  DISP=(MOD,PASS),DSN=&&LOADSET,
//               UNIT=SYSDA,DCB=BLKSIZE=3120,
//               SPACE=(TRK,(10,5))
//SYSLIB     DD  DISP=(NEW,DELETE),DSN=&&SYSLIB,
//               UNIT=SYSDA,DCB=BLKSIZE=80,
//               SPACE=(TRK,(1,1,1))
//SYSUT1     DD  DSN=&&SYSUT1,UNIT=SYSDA,
//               SPACE=(TRK,(10,5))
//SYSPUNCH   DD  DUMMY
//SYSPRINT   DD  SYSOUT=*
```

```
//*********************************************************************
//*
//*            STEP 6:  Link edit the static command program
//*
//*********************************************************************
//LINK      EXEC PGM=IEWL,COND=(0,NE,ASMCMD) ,
//               PARM='LET,LIST,CALL,RENT,MAP'
//SYSLIB     DD  DISP=SHR,DSN=&DB2LIB
//SYSLIN     DD  DISP=(OLD,DELETE),DSN=&&LOADSET
//SYSLMOD    DD  DISP=SHR,DSN=&EZTPPGM(&STCPGMN)
//SYSUT1     DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT   DD  SYSOUT=*
//*********************************************************************
//*
//*            STEP 7:  Bind and grant authorization to the
//*                     static command program's application plan
//*
//*********************************************************************
//AUTH      EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(0,NE,ASMCMD)
//STEPLIB    DD  DISP=SHR,DSN=&RUNLIB
//           DD  DISP=SHR,DSN=&DB2MLIB
//DBRMLIB    DD  DISP=SHR,DSN=&DBRMLIB
//SYSPRINT   DD  SYSYOUT=*
//SYSTSPRT   DD  SYSYOUT=*
//SYSOUT     DD  SYSYOUT=*
//REPORT     DD  SYSYOUT=*
//*
//          PEND
```

Execution JCL

The sample execution JCL is installed as member DB2BINDX into your SAMPJCL dataset. DB2BINDX executes the EZTPDB2 procedure shown on the previous pages. Step EXTPGO requires you to provide values for the program name to execute and the required libraries needed for STEPLIB.

```
//jobname JOB accounting.info,USER=userid,COND=(8,LT)
//*********************************************************************
//*
//*     STEP A:  Execute the EZTPDB2 procedure
//*
//*********************************************************************
//EZTPDB2  EXEC EZTPDB2,
//              DBRMLIB='your.dbrmlib'
//              DB2LIB='your.DB2.loadlib',
//              RUNLIB='your.DB2.runlib',
//              EZTPLIB='your.eztp.loadlib',
//              EZTPPGM='your.linked.eztpgm.loadlib,
//              PSQLLIB='your.pansql.loadlib',
//              DBRMNME='dbrmname',
//              STCPGMN='static.command.program.name'
//*********************************************************************
//*
//*   DD statements for STEP 1 - EZT Compile
//*
//*********************************************************************
//EZTPLUS.SYSIN     DD   *
PARM LINK(eztpgm)  PLAN (planname stcpgmn) BIND STATIC-ONLY ...
    ... CA-Easytrieve Plus source statements ...
/*
```

```
//**********************************************************************
//*
//*   DD statements for STEP 7 - - Bind and Grant
//*
//**********************************************************************
//AUTH.SYSTSIN   DD *
 DSN SYSTEM(ssid)
 BIND PLAN(planname)  MEMBER(dbrmname) -
      ACT(REPLACE) ISOLATION(CS)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIADs)
 END
/*
//AUTH.SYSIN     DD *
 GRANT EXECUTE  ON PLAN planname TO PUBLIC;
/*
//**********************************************************************
//*
//*          STEP B:  Execute the linked EZTP program
//*
//**********************************************************************
//EZTPGO  EXEC PGM=eztpgm
//STEPLIB  DD   DSN=your.EZTPLIB.dataset,DISP=SHR
//         DD   DSN=your.EZTPPGM.dataset,DISP=SHR
//         DD   DSN=your.PSQLLIB.dataset,DISP=SHR
//         DD   DSN=your.DB2LIB.dataset,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//EZTVFM   DD  UNIT=SYSDA,SPACE=(CYL,(1,5))
// ... Other DD statements as needed by the program
```

Procedure Overrides

Procedure	Action
DBRMLIB	Must name a library to contain the DBRM for the CA-Easytrieve Plus static application plan.
DB2LIB	Names your IBM DB2 *load library* that contains the program DSNHPC.
RUNLIB	Names your IBM DB2 *runlib* containing the program DSNTIAD.
EZTPLIB	Names the CA-Easytrieve Plus *loadlib*.
EZTPPGM	Names a *user loadlib* to contain the linked CA-Easytrieve Plus program and the static command program.
PSQLLIB	Names your Pan/SQL *load library*.

Procedure	Action
DBRMNME	Identifies the DBRM.
STCPGMN	Identifies the generated static command program. The name must match the *stcpgmn* specified on the PARM statement in your CA-Easytrieve Plus program source. If no *stcpgmn* is specified on the PARM statement, STCPGMN must be the same as the *planname* specified on the PARM statement.

JCL Modifications

AUTH.SYSTSIN binds the STATIC SQL program plan.

- Change *ssid* to reference the default DB2 for OS/390 and z/OS subsystem you are using.
- Change *planname* to be the same value you specified on the PARM statement.
- Change *dbrmnme* to be the same value you specified on the DBRMNME parameter of the procedure overrides.
- Change DSNTIADs to reference the correct PLAN for the DB2 for OS/390 and z/OS utility program DSNTIAD.

AUTH.SYSIN grants access to the static SQL program plan to PUBLIC. Change *planname* to be the same as that specified on the BIND for AUTH.SYSTSIN.

Step DP2PRE ends with a return code of 4. This is a normal operation.

Mixed IMS and DB2 for OS/390 and z/OS Execution

If your CA-Easytrieve Plus program accesses an IMS database and DB2 for OS/390 and z/OS database, you should run your program under the control of DB2-DL/I BATCH SUPPORT.

This ensures synchronization of checkpoints and rollbacks across both IMS and DB2 for OS/390 and z/OS.

Under this control, SQL COMMITs and SQL ROLLBACKs are not permitted.

To do this, change your execution JCL to execute Module DSNMTV01 instead of EZTPA00 or your linked application program. DSNMTV01 then loads the real program.

The following is sample JCL:

```
//EZIMSDB2 EXEC  PGM=DFSRRC00,
//         PARM='DLI,DSNMTV01,PSBNAME,,,,,,,,,,O,N,N,'
//STEPLIB  DD  DSN=IMSVS.RESLIB,DISP=SHR
//         DD  DSN=IMSVS.PGMLIB,DISP=SHR
//         DD  DSN=your.ibm.db2.sspgm.library,DISP=SHR
//         DD  DSN=your.eztp.loadlib,DISP=SHR
//         DD  DSN=your.pansql.loadlib,DISP=SHR
//IMS      DD  DSN=IMSVS.PSBLIB,DISP=SHR
//         DD  DSN=IMSVS.DRDLIB,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//EZTVFM   DD  UNIT=SYSDA,SPACE=(4096,(100,100))
//DDOTV02  DD  DSN=&&TEMP,DISP=(NEW,PASS),SPACE=(TRK,(5,5),
                  UNIT=SYSDA,DCB=(RECFM=VB,LRECL=4092,BLKSIZE=4096)
//DDITV02  DD  *
  SSN,LIT,ESMT,RTT,ERR,CRC,CONNECTION_NAME,PLAN,PROG
/*
//SYSIN    DD  *
    CA-Easytrieve Plus statements follow
     .
     .
     .
/*
//
```

Note: Under the control of DB2 for OS/390 and z/OS or DLI batch support, your STEPLIB must define the IMS RESLIB before the DB2 for OS/390 and z/OS library.

The DDITV02 DD defines the following parameters:

Parameter	Description
SSN	DB2 for OS/390 and z/OS subsystem
LIT	Language Interface token value
ESMT	Initialization module name, must be DSNMIN10
RTT	Resource Translation Table
ERR	Region Error Option
CRC	Command Recognition Character
CONNECTION_NAME	A unique one-to eight-character name
PLAN	DB2 for OS/390 and z/OS plan name
PROG	Application program name to execute—either EZTPA00 or linked program name.

Refer to your IBM DB2 guides for a detailed explanation of these parameters.

DB2 for VSE Execution

The following example illustrates the JCL necessary to execute CA-Easytrieve Plus with DB2 for VSE.

VSE JCL

```
* $$ JOB JNM=jobname
// JOB     jobname
// UPSI    b
// DLBL    EZTP,'your.eztp.library'
// EXTENT  ,volser
// ASSGN   SYS001,DISK,VOL=volser,SHR
// DLBL    SORTWK1,,0
// EXTENT  SYS001,volser,,,start,lgth
// ASSGN   SYS010,DISK,VOL=volser,SHR
// DLBL    EZTVFM,,0,SD
// EXTENT  SYS010,volser,,,start,lgth
// DLBL SQLLIB,'your.pansql.library',0,SD
// EXTENT  ,volser
// LIBDEF  PHASE,SEARCH=(SQLLIB.sublib,EZTP.sublib)
// EXEC    EZTPA00,SIZE=512K
PARM USERID('user-id' 'password')
 ... CA-Easytrieve Plus SQL/DS source statements ...
/*
* $$ EOJ
```

Note: When executing a CA-Easytrieve Plus program withDB2 for VSE, no JCL changes are required for Multiple User Mode. For the Single User Mode of DB2 for VSE, you should specify the SIZE=(AUTO,*xxx*K) parameter on your JCL EXEC statement, where *xxx*K is the amount of storage required for normal CA-Easytrieve Plus execution. The *xxx*K value is usually between 256K and 512K.

CA-IDMS/SQL Execution

The following example illustrates the OS/390 and z/OS JCL to execute CA-Easytrieve Plus with IDMS/SQL under central version.

OS/390 and z/OS JCL

```
//jobname  JOB accounting.info
//stepname EXEC PGM=EZTPA00
//STEPLIB  DD   DISP=SHR,DSN=your.eztp.loadlib
//         DD   DISP=SHR,DSN=your.idms.loadlib
//         DD   DISP=SHR,DSN=pansql.idms.loadlib
//SYSPRINT DD   SYSOUT=A
//SYSOUT   DD   SYSOUT=A
//SORTWK01 DD   UNIT=SYSDA,SPACE=(CYL,1)
//SYSCTL   DD   DISP=SHR,DSN=idms.sysctl
//SYSIDMS  DD   DISP=SHR,DSN=your.sysidms.parm.file
//EZTVFM   DD   UNIT=SYSDA,SPACE=(4096,(100,100))
//SYSIN    DD   *
 ...CA-Easytrieve Plus source statements ...
/*
```

Note: To run in local mode, you must make the CA-IDMS dictionary and database available to CA-Easytrieve Plus through IDMS Dynamic Allocation services by defining them in the DMCL or through DD statements in the JCL.

VSE JCL

The following example illustrates the VSE JCL to execute CA-Easytrieve Plus with IDMS/SQL under central version.

```
*  $$ JOB JNM=jobname
// JOB      jobname
// UPSI     b
// DLBL     EZTP,'your.eztp.library'
// EXTENT   ,volser
// DLBL     IDMS,'idms.library'
// EXTENT   ,volser
// DLBL     SQLLIB,'your.pansql.library'
// EXTENT   ,volser
// DLBL     SYSIDMS,'your.idms.sysidms.file'
// EXTENT   ,volser
// DLBL     SYSCTL,'your idms.sysctl.file'
// EXTENT   ,volser
// LIBDEF   PHASE,SEARCH=(EZTP.sublib,SQLLIB.sublib,IDMS.sublib)
// ASSGN    SYS001,DISK,VOL=volser,SHR
// DLBL     SORTWK1,,0
// EXTENT   SYS001,volser,,,start,length
// DLBL     EZTVFM,,0,SD
// EXTENT   SYS010,volser,,,start,length
// EXEC     EZTPA00
 ... CA-Easytrieve Plus source statements ...
```

Note: To run in local mode, you must make the CA-IDMS dictionary and database file available to CA-Easytrieve Plus. This can be done either through IDMS dynamic allocation service in the DMCL or by specifying DLBLs in the JCL.

CA-Datacom/SQL Execution

The following example illustrates the JCL necessary to compile and go with CA-Datacom/SQL.

OS/390 and z/OS JCL

```
//jobname   JOB accounting.info
//stepname  EXEC  PGM=EZTPA00,REGION=512K
//STEPLIB   DD DISP=SHR,DSN=your.eztp.loadlib
//          DD DISP=SHR,DSN=your.pansql.loadlib
//          DD DISP=SHR,DSN=your.datacom.CUSLIB
//          DD DISP=SHR,DSN=your.datacom.CAILIB
//SYSPRINT  DD SYSOUT=A
//SYSSNAP   DD SYSOUT=A
//SYSOUT    DD SYSOUT=A
//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,1)
//EZTVFM    DD UNIT=SYSDA,SPACE=(4096,(100,100))
//userfile  DD dd-parms
//SYSIN     DD *
    ... CA-Easytrieve Plus source statements ...
```

VSE JCL Examples

The following example illustrates the JCL necessary to compile and go with CA-Datacom/SQL.

```
*  $$ JOB JNM=jobname
//  JOB jobname
//  DLBL EZTP,'your.eztp.library,0,SD
//  EXTENT SYS003,volser,1,0,start,lgth
//  ASSGN SYS003,nnn
//  DLBL   SQLLIB,'your.pansql.library'
//  EXTENT ,volser
//  DLBL   DATACOM,'your.datacom.library'
//  EXTENT ,volser
//  LIBDEF *,SEARCH(EZTP.sublib,SQLLIB.sublib,DATACOM.sublib)
//  ASSGN  SYSOUT,...
//  ASSGN  SYS010,...
//  ASSGN  SYS008,...
//  DLBL   SORTWK1,,0,DA
//  EXTENT SYS001,volser,,,start,lgth
//  DLBL   EZTVFM,,0,SD
//  EXTENT SYS010,volser,,,start,lgth
//  DLBL   INREC,,0,SD
//  EXTENT SYS008,volser,,,start,lgth
//  EXEC   EZTPA00,SIZE=512K
    CA-Easytrieve Plus source statement
/*
/&
* $$ EOJ
```

ORACLE Execution

The following JCL illustrates how to run a CA-Easytrieve program with ORACLE.

ORACLE JCL

```
//jobname   JOB accounting.info
//stepname  EXEC  PGM-EZTPA00,REGION-512K
//STEPLIB   DD DISP=SHR,DSN=your.eztp.loadlib
//          DD DISP=SHR,DSN=your.oracle.pansql.loadlib
//          DD DISP=SHR,DSN=your.oracle.SQLLIB
//          DD DISP=SHR,DSN=your.oracle.CMDLOAD
//SYSPRINT  DD SYSOUT=A
//SYSSNAP   DD SYSOUT=A
//SYSOUT    DD SYSOUT=A
//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,1)
//EZTVFM    DD UNIT=SYSDA,SPACE=(4096,(100,100))
//userfile  DD dd-parms
//SYSIN     DD *
    CA-Easytrieve Plus source statements
```

Chapter 7

Examples

The following examples illustrate the functions of the various SQL statements.

For information concerning %SQLTABLE, see the Sample Database section in the "Library Section Definition" chapter.

Automatic Retrieval

Automatic Retrieval: All Columns

The code in the following example retrieves all columns and all rows from the PERSONNEL table. A report is generated showing WORKDEPT, EMPNAME, and EMPPHONE. The report is sorted by WORKDEPT.

```
%SQLTABLE .
JOB INPUT SQL
   SELECT * FROM PERSONNEL                            +
      INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE
   IF NULLPHONE < 0 .* PHONE PRESENT?
      EMPPHONE = 0  .* NO, SET TO 0
   END-IF
   PRINT PERSNL
REPORT PERSNL LINESIZE 65
 SEQUENCE WORKDEPT
 LINE WORKDEPT EMPNAME EMPPHONE
```

Automatic Retrieval: Selected Columns

The code in the following example retrieves all rows of selected columns from the PERSONNEL table and generates a report showing WORKDEPT and EMPNAME. SQL orders the rows by WORKDEPT.

```
%SQLTABLE .
JOB INPUT SQL
   SELECT EMPNAME, WORKDEPT     +
      FROM PERSONNEL            +
      ORDER BY WORKDEPT         +
      INTO :EMPNAME, :WORKDEPT
   PRINT PERSNL
REPORT PERSNL LINESIZE 65
 LINE WORKDEPT EMPNAME
```

Automatic Retrieval: Multiple Tables

The code in the following example retrieves an employee name and the corresponding department name from the PERSONNEL and DEPARTMENTS tables. The PARM statement parameters are needed only for DB2 for VSE processing.

```
PARM USERID('SQLDBA' 'SQLDBAPW') +
     PREPNAME(EASYPLUS 'SQLDBA')
%SQLTABLE .
JOB INPUT SQL
   SELECT EMPNAME, DEPTNAME          +
      FROM PERSONNEL, DEPARTMENTS    +
      WHERE  WORKDEPT = DEPTNUMBER   +
      INTO :EMPNAME, :DEPTNAME
   PRINT PERSNL
REPORT PERSNL LINESIZE 65
 LINE EMPNAME DEPTNAME
```

Automatic Retrieval: SQL FILE

The following example demonstrates the use of an SQL file as automatic input in a JOB activity. Also, it demonstrates the use of the SQL catalog INCLUDE facility.

Note: The host variables on the select-clause cannot reference the table name.

```
DEFINE NULLPHONE W  2 B 0
FILE FILES SQL                               +
          (SELECT * FROM PERSONNEL           +
           INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE)
SQL INCLUDE LOCATION * FROM PERSONNEL
JOB INPUT FILES
 IF NULLPHONE < 0
    EMPPHONE = 0
 END-IF
 PRINT PERSNL
REPORT PERSNL LINESIZE 65
 SEQUENCE WORKDEPT
 LINE WORKDEPT EMPNAME EMPPHONE
```

Automatic Retrieval: SQL FILE

The following example demonstrates the use of an SQL file as automatic input in a JOB activity. The select-clause is not coded on the FILE statement. The SELECT statement must be coded immediately following the JOB statement. In this example, the select-clause can reference the table name as a host variable.

Note: The DEPTNAME field is a VARYING field.

```
FILE FILES SQL
 SQL INCLUDE LOCATION * FROM DEPARTMENTS
JOB INPUT FILES
 SELECT * FROM DEPARTMENTS              +
           INTO :DEPARTMENTS
 PRINT DEPT
REPORT DEPT LINESIZE 65
 SEQUENCE DEPTNUMBER
 LINE DEPTNUMBER DEPTNAME
```

Automatic Retrieval: SQL FILE Using SQL Functions

The following example shows using the SQL-supplied functions COUNT, AVG and SUM to produce information for use in the program.

```
 FILE   FILES   SQL
  SQL  INCLUDE  LOCATION *  FROM  PERSONNEL
*
 NUM-IN-DEPT   W  4  P  0
 AVG-SALARY    W  4  P  2
 TOT-SALARY    W  4  P  2
*
JOB INPUT  FILES
   SELECT  WORKDEPT, COUNT(*), AVG(SALARY), SUM(SALARY) +
    FROM   PERSONNEL                                    +
    GROUP   BY WORKDEPT                                 +
    HAVING  DEPT < 80                               +
    INTO   :WORKDEPT, :NUM-IN-DEPT, :AVG-SALARY, :TOT-SALARY
    ...
```

Automatic Retrieval: SQL FILE With NULL Indicators

The following example shows how to test for null indicators with the SQL INCLUDE statement. An indicator array is coded following the SQL INCLUDE. The indicator array occurs four times because there are 4 columns being defined through the SQL INCLUDE. In this case, only the third field is tested for NULL.

```
FILE    FILES       SQL
  SQL  INCLUDE   LOCATION *  FROM   PERSONNEL
*
INDARRAY  W  2   B  0  OCCURS 4.    * Null indicator array
*
JOB INPUT  FILES
  SELECT *  FROM    PERSONNEL +
  INTO :PERSONNEL  :INDARRAY
  IF INDARRAY(3)   <  0.         * Is the phone number(3rd column)null?
    ...
```

Automatic Retrieval: SQL FILE With Synchronized File Processing

The following example demonstrates using synchronized file processing to match a selected group of SQL rows to a flat file.

```
FILE PERSNL
 EMP#     9   5  A
 NAME    17  16  A
*
FILE PERSDB2 SQL  +
 (SELECT DISTINCT NAME, EMPNO, REGION, ADDRSTREET   +
        FROM DEMO.PERSNL                            +
        WHERE REGION IN ('2')                       +
        ORDER BY EMPNO                              +
        INTO :NAME, :EMPNO, :REGION, :ADDRSTREET)
   SQL INCLUDE LOCATION * FROM DEMO.PERSNL
*
 JOB  INPUT  (PERSNL       KEY (EMP#)               +
              PERSDB2       KEY (EMPNO))
*
      IF MATCHED
        ...
```

Note: *SQL INCLUDE LOCATION ** must be coded after *SELECT* with Synchronized File Processing.

Controlled Retrieval

Native SQL: All Columns

The code in the following example retrieves all columns from the PERSONNEL table. The PARM statement parameters are needed only for DB2 for VSE processing.

```
PARM USERID('SQLDBA' 'SQLDBAPW') +
     PREPNAME(EASYPLUS 'SQLDBA')
%SQLTABLE .
SQL DECLARE CURSOR1 CURSOR FOR    +
SELECT *                          +
   FROM PERSONNEL
JOB INPUT NULL
* BELOW FOR SQL/DS ONLY
   SQL CONNECT :SYS-USERID IDENTIFIED BY :PASSWORD
   PERFORM CHECKSQL
* ABOVE FOR SQL/DS ONLY
   SQL OPEN CURSOR1
   PERFORM CHECKSQL
   DO WHILE SQLCODE NE 100
      SQL FETCH CURSOR1              +
          INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE
      PERFORM CHECKSQL
      IF NULLPHONE < 0    .* PHONE PRESENT?
         EMPPHONE = 0     .* NO, SET TO 0
      END-IF
      IF SQLCODE NE 100   .* NOT END OF TABLE
         PRINT PERSNL
      END-IF
   END-DO
   SQL CLOSE CURSOR1
   PERFORM CHECKSQL
   STOP
CHECKSQL.PROC
 IF SQLCODE NE 0 AND SQLCODE NE 100
    DISPLAY 'SQLCODE = ' SQLCODE
    STOP EXECUTE
 END-IF
END-PROC
REPORT PERSNL LINESIZE 65
 LINE EMPNAME WORKDEPT EMPPHONE
```

Native SQL: Reassign Departments

The code in the following example reassigns all employees in department 901 to department 109 and displays the names of those employees involved. The PARM SSID is necessary only if you want to access a DB2 for OS/390 and z/OS subsystem other than the default.

```
PARM SSID('DB2B')
%SQLTABLE .
SQL DECLARE CURSOR1 CURSOR FOR    +
SELECT EMPNAME                    +
   FROM PERSONNEL                 +
   WHERE WORKDEPT = 901           +
   FOR UPDATE OF WORKDEPT
JOB INPUT NULL
   SQL OPEN CURSOR1
   PERFORM CHECKSQL
   DO WHILE SQLCODE NE 100
      SQL FETCH CURSOR1           +
          INTO :EMPNAME
      PERFORM CHECKSQL
      IF SQLCODE NE 100   .* NOT END OF TABLE
         PRINT PERSNL
         SQL UPDATE PERSONNEL                 +
             SET WORKDEPT = 109               +
             WHERE CURRENT OF CURSOR1
         PERFORM CHECKSQL
      END-IF
   END-DO
   SQL CLOSE CURSOR1
   PERFORM CHECKSQL
   STOP
CHECKSQL.PROC
 IF SQLCODE NE 0 AND SQLCODE NE 100
    DISPLAY 'SQLCODE = ' SQLCODE
    STOP EXECUTE
 END-IF
END-PROC
REPORT PERSNL LINESIZE 65
 LINE EMPNAME
```

Native SQL: Update Phone Numbers

The following example illustrates the installation of a new phone system. All phone numbers have become five digits. The first character has become seven (7) and the remaining digits are the same as the old phone system.

If an employee did not previously have a phone number, his record is not updated. No update report is necessary.

```
JOB INPUT NULL
    SQL UPDATE PERSONNEL                +
       SET EMPPHONE = 70000 + EMPPHONE  +
       WHERE EMPPHONE IS NOT NULL
    PERFORM CHECKSQL
    STOP
CHECKSQL. PROC
 IF SQLCODE NE 0 AND SQLCODE NE 100
    DISPLAY 'SQLCODE = ' SQLCODE
    STOP EXECUTE
 END-IF
END-PROC
```

Native SQL: SQL FILE

The following example demonstrates the use of an SQL file as controlled input.

Note: The SQL FILE has the options of DEFER and DBCSCODE specified.

The SQL cursor (that is automatically defined) is not opened until the first GET statement is issued, thereby permitting the host variable in the into clause of the select-clause to initialize.

Note: The no-more-rows condition (EOF) could have been tested in various ways.

```
DEFINE DEPTNO W 3 P 0
FILE FILES DEFER DBCSCODE IBM                +
           SQL                               +
          (SELECT * FROM PERSONNEL           +
           WHERE DEPTNUMBER > :DEPTNO        +
           INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE)
SQL INCLUDE LOCATION * FROM PERSONNEL START START-UP
JOB INPUT NULL
 GET FILES
 IF EOF FILES     . *COULD HAVE BEEN: IF SQLCODE = 100
                    *OR               IF FILES:FILE-STATUS = 100
    STOP
 END-IF
 IF NULLPHONE < 0
    EMPPHONE = 0
 END-IF
 PRINT PERSNL
START-UP. PROC
 DEPTNO = 945
END-PROC
REPORT PERSNL LINESIZE 65
 SEQUENCE WORKDEPT
 LINE WORKDEPT EMPNAME EMPPHONE
```

Index